Session 208

# Audio Services in Mac OS X

Craig Keithley
USB & FireWire Technology Manager
Worldwide Developer Relations

Session 208

# Audio Services in Mac OS X

Jeff Moore, Doug Wyatt, William Stewart
Core Audio

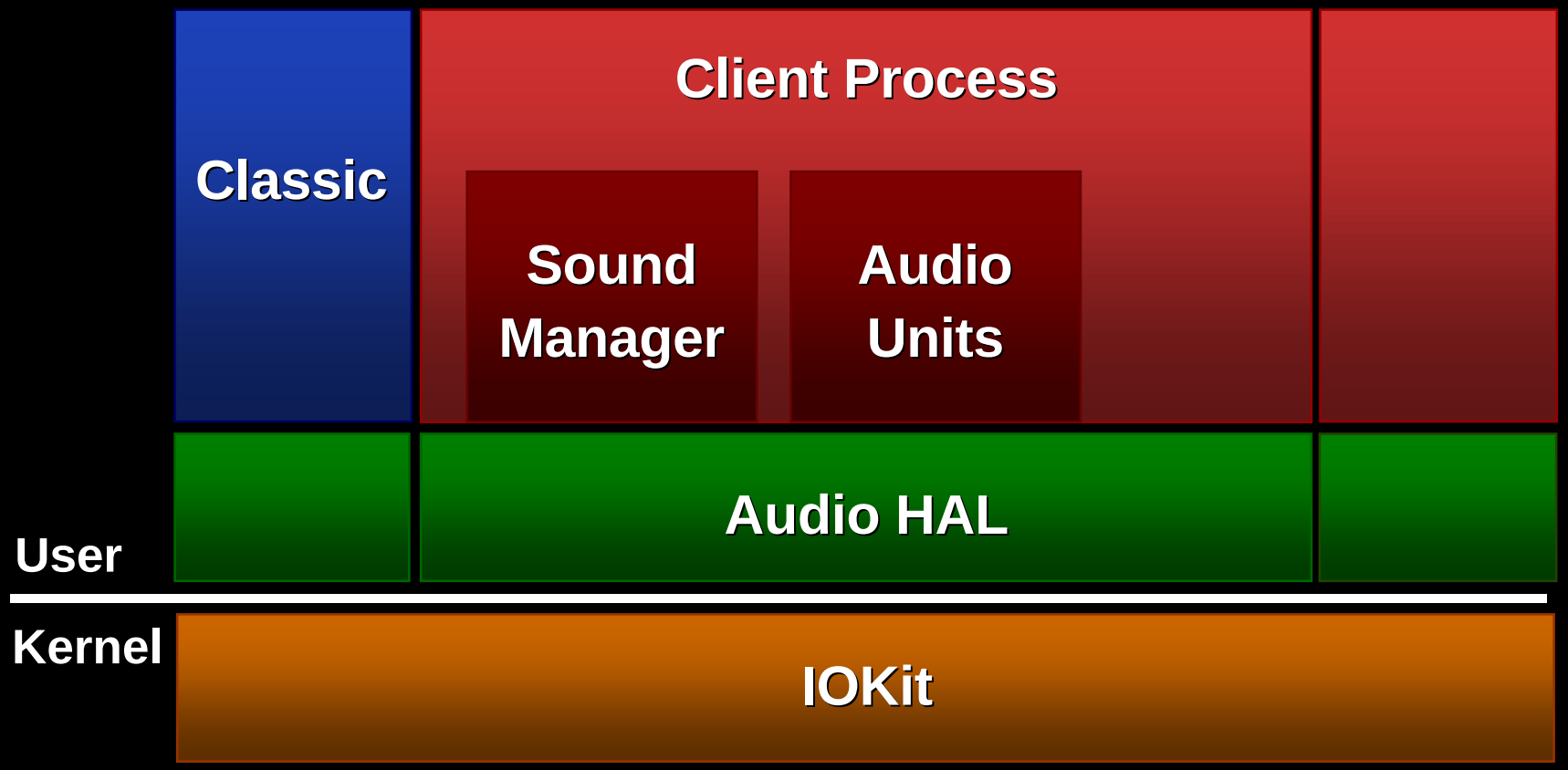
# Core Audio Features

- Flexible audio format
  - Multi-channel audio I/O
  - Both PCM and non-PCM formats
  - Float32 is generic format
  - Fully specifiable sample rates

# Core Audio Features

- Multiple application usage of audio devices
  - Application determines latency
- Prevalence of timing information
- Both C and Java APIs

# Core Audio Architecture



# Core Audio Architecture

- CoreAudio.framework
  - AudioDevice interface
  - Presents audio hardware to applications
  - I/O Cycle

# Core Audio Architecture

- AudioToolbox.framework
  - AUGraph
  - MusicSequence
  - MusicPlayer
- AudioUnits.framework
  - Processing unit components
  - MusicDevice
  - Default output unit

# Carbon Sound

- SoundManager carbonized
  - Near equivalent API to Classic SM
  - Deprecated APIs removed
  - Classic
    - SM is global
    - Provides hardware model
  - Mac OS X
    - SM is per-process
    - Hardware model provided by CoreAudio

# Core MIDI Architecture

- Provides complete representation of MIDI devices
- Virtual end-points for Inter-Application communication of MIDI information
- Same time-stamping structures as Core Audio frameworks

# Devices—What Devices?

- Apple currently supports audio drivers for:
  - Built-in hardware
  - USB Audio-class driver
    - Complete implementation of spec
- FireWire Audio
  - 61883-6 transport layer
  - mLAN connection management

# Devices—What Devices?

- Apple currently supports MIDI drivers for:
  - USB
    - MIDI-class Driver
    - Base class for vendor-specific USB drivers
- FireWire
  - 61883-6 transport protocol
  - mLAN Connection Management

# Audio HAL

**Jeff Moore**
**Engineer, Core Audio**

# Introduction

- The Audio Hardware Abstraction Layer provides the lowest user level access to the audio devices attached to the system
  - Efficiently moves data to/from the device across the User/Kernel boundary
  - Manipulates device attributes (volume, mute, sample rate, etc.)
  - Provides synchronization information

# Audio HAL Design Goals

- A clean, low overhead signal path
- Multiple simultaneous clients
- Support more channels, higher bit depths and higher sample rates
- Support solid synchronization primitives
- Small, but complete API

# Audio HAL API Basics

- Object-oriented C with Java interfaces
- Two kinds of objects
  - Device objects (represented as AudioDeviceIDs) encapsulate a single audio device
  - While not strictly an object, the global context provides access to system or process wide constructs

# Audio HAL API Basics

- Every object has properties that describe and manipulate aspects of the device
  - Name, buffer size, volume of channel 2, etc.
  - Unique IDs with arbitrary data
  - Some are read-only
  - Notifications available for when a property's value changes

# Global Properties

- Routines/constants use the prefix, **AudioHardware**
- Important properties
  - **kAudioHardwarePropertyDevices**
  - **kAudioHardwarePropertyDefaultInputDevice**
  - **kAudioHardwarePropertyDefaultOutputDevice**
  - **kAudioHardwarePropertySleepingIsAllowed**

# Code Example

- Getting the list of devices (step 1)

```
//    get the size of the list
UInt32 theSize;
theStatus = AudioHardwareGetPropertyInfo (
                    kAudioHardwarePropertyDevices,
                    &theSize, NULL );

theNumberDevices = theSize / sizeof(AudioDeviceID);
```

# Code Example

- Getting the list of devices (step 2)

```
//    allocate enough space to hold the list
theDeviceList = (AudioDeviceID*) malloc (
            theNumberDevices * sizeof(AudioDeviceID) );
```

# Code Example

- Getting the list of devices (step 3)

```
//    get the device list
theSize = theNumberDevices *
                    sizeof(AudioDeviceID);

theStatus = AudioHardwareGetProperty (
                    kAudioHardwarePropertyDevices,
                    &theSize, theDeviceList )
```

# Audio Devices

- Routines/constants use the prefix, AudioDevice
- When querying a device's properties, it is important to be as specific as possible
- Important properties
  - **kAudioDevicePropertyBufferSize**
  - **kAudioDevicePropertyStreamConfiguration**
  - **kAudioDevicePropertyVolumeScalar**

# Code Example

- Setting the volume of channel 2 (step 1)

```
//    find out if output channel 2 has a volume control
Boolean   isWritable;
theStatus = AudioDeviceGetPropertyInfo (
                    theDeviceID, 2, false,
                    kAudioDevicePropertyVolumeScalar,
                    NULL, &isWritable );
```

# Code Example

- Setting the volume of a channel 2 (step 2)

```
//    if it does, set the value
If ( (theStatus == kAudioHardwareNoError)
                    && isWritable) {
    Float32 theValue = 1.0;
    theStatus = AudioDeviceSetProperty (
                    theDevice, NULL, 2, false,
                    kAudioDevicePropertyVolumeScalar,
                    sizeof(Float32), &theValue );
}
```

# I/O Theory of Operation

- I/O was re-thought to take advantage of what Mac OS X does well
- Data is passed to/from the hardware via a ring buffer
  - The ring buffer is on the order of 3/4ths of a second in size
  - The ring buffer is shared across the entire system

# I/O Theory of Operation

- The hardware's progress through the ring buffer is tracked by time stamping the wrap point
  - These "zero" time stamps drive the HAL's interpolation engine
  - Hardware interrupts only occur at the wrap point
  - Much less taxing on the system

# I/O Threads

- Each device has a single thread in each process in which to call all the registered IOProcs
- This thread is a mach real time thread to ensure the highest degree of timing accuracy
- The HAL uses a lower priority thread for notifications
  - Use kAudioHardwarePropertyRunLoop to specify your own thread

# I/O Thread Algorithm

- The pseudo-code for the I/O thread (part 1)

```
while(not done)
{
    Calculate next time out;
    Sleep until next time out;
```

# I/O Thread Algorithm

- The pseudo-code for the I/O thread (part 2)

```
if(device has input)
{
    Calculate input time;
    Call driver to copy data to IOProcs'
        input AudioBufferList;
}
```

# I/O Thread Algorithm

- The pseudo-code for the I/O thread (part 3)

```
if(device has output)
{
    Calculate output time;
    Clear IOProcs' output AudioBufferLists;
}

Call IOProcs;
```

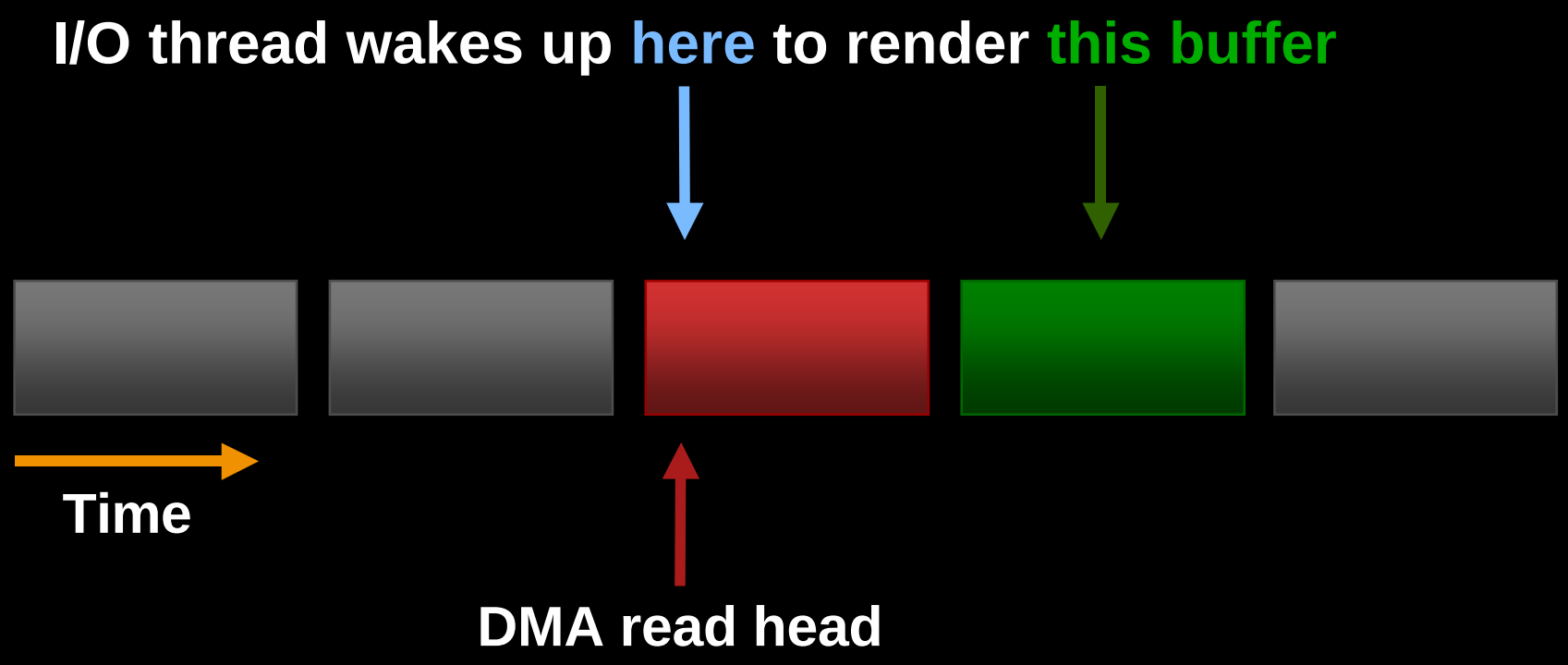
# I/O Thread Algorithm

- The pseudo-code for the I/O thread (part 4)

```
    if(device has output)
    {
        Call driver to mix IOProcs' output
            AudioBufferLists to the system mix bus;
    }

    Handle any deferred operations;
}
```

# I/O Thread Timing



# IOProcs

- A client registers an IOProc with a device it wishes to do I/O
- IOProcs can be started and stopped
  - The device starts up when the first IOProc in the system is started
  - The device stops when the last IOProc in the system is stopped

# IOProcs

- When an IOProc is called, it receives all the input data and is provided a buffer in which to write the ouput data
- Three timestamps are provided
  - When the input data was acquired
  - When the output data will be played
  - The current time

# IOProcs

- IOProcs are passed data using an **AudioBufferList** data structure
- An **AudioBufferList** has an **AudioBuffer** for each stream the device has
- To find out ahead of time what the **AudioBufferList** will look like, query the **kAudioDevicePropertyStreamConfiguration** property

# Code Example

- Writing an IOProc (Part 1)

```
OSStatus MyEngine::IOProc (
    AudioDeviceID                   inDevice,
    const AudioTimeStamp*           inNow,
    const AudioBufferList*          inInputData,
    const AudioTimeStamp*           inInputTime,
    AudioBufferList*                outOutputData,
    const AudioTimeStamp*           inOutputTime,
    void*                           inClientData )
{
```

# Code Example

- Writing an IOProc (Part 2)

```
//    For this IOProc, inClientData is actually
//    a pointer to a C++ object of type MyEngine
MyEngine* theEngine = (MyEngine*) inClientData;
```

# Code Example

- Writing an IOProc (Part 3)

```
    if ( inInputData != NULL ) {
        theEngine->ProcessInputData ( inInputData,
                                      inInputTime);
    }
    if ( outOutputData != NULL ) {
        theEngine->RenderOutputData ( outOutputData,
                                      inOutputTime );
    }
}
```

# Code Example

- Installing and removing an IOProc

```
//    Install MyEngine::IOProc
theStatus = AudioDeviceAddIOProc ( myDevice,
                    MyEngine::IOProc, myEngineObject);

//    Remove MyEngine::IOProc
theStatus = AudioDeviceRemoveIOProc ( myDevice,
                    MyEngine::IOProc);
```

# Code Example

- Starting and stopping an IOProc

```
//    Start MyEngine::IOProc
theStatus = AudioDeviceStart ( myDevice,
                    MyEngine::IOProc );

//    Stop MyEngine::IOProc
theStatus = AudioDeviceStop ( myDevice,
                    MyEngine::IOProc );
```

# Default Output Unit

**Doug Wyatt**
**Engineer, Core Audio**

# Default Output Unit

- Higher-level API to HAL
- Handles HAL notifications
- Provides optional sample rate conversion

# Default Output Unit Example

```
#include < AudioToolbox/DefaultAudioOutput.h >

// open the component and initialize it
AudioUnit outputUnit;
OSStatus err;

err = OpenDefaultAudioOutput ( &outputUnit );

err = AudioUnitInitialize ( outputUnit );
```

# Default Output Unit Example

```
// find the HAL device

AudioDeviceID device = NULL;

size = sizeof(device);
AudioUnitGetProperty(
    outputUnit,
    kAudioOutputUnitProperty_CurrentDevice,
    kAudioUnitScope_Global,
    0,
    &device,
    &size);
```

# Default Output Unit Example

```
// set the device's buffer size

UInt32 bufferSize = 64 * nChannels * sizeof (Float32);

AudioDeviceSetProperty ( device, NULL, 0, 0,
    kAudioDevicePropertyBufferSize,
    sizeof ( UInt32 ),
    &bufferSize );
```

# Default Output Unit Example

```
Float64 gSampleRate;

// set our rendering sample rate

gSampleRate = 22050.0;
AudioUnitSetProperty(
        outputUnit,
        kAudioUnitProperty_SampleRate,
        kAudioUnitScope_Input, // note input scope!
        0,
        &gSampleRate,
        sizeof ( gSampleRate ) ) ;
```

# Default Output Unit Example

```
// set our rendering callback function

AudioUnitInputCallback input;

input.inputProc = MyRenderer;
input.inputProcRefCon = NULL;

AudioUnitSetProperty (
    outputUnit,
    kAudioUnitProperty_SetInputCallback,
    kAudioUnitScope_Global, 0,
    &input, sizeof ( input ) ) ;
```

# Default Output Unit Example

```
// rendering function
OSStatus    MyRenderer ( void *inRefCon,
        AudioUnitRenderActionFlags inActionFlags,
        const AudioTimeStamp *inTimeStamp,
        UInt32 inBusNumber,
        AudioBuffer *ioData )
{
        int nChannels = ioData->mNumberChannels;
        UInt32 nFrames = ioData->mDataByteSize /
                (sizeof(Float32) * nChannels);
        Float32 *samplePtr = (Float32 *) ioData->mData;
        // … write nFrames * nChannels Float32's into
        //          samplePtr array ...
}
```

# Default Output Unit Example

```
// start sound
AudioOutputUnitStart ( outputUnit ) ;

// …render proc starts getting called, sound plays ...

// stop sound
AudioOutputUnitStop ( outputUnit ) ;

// clean up
CloseComponent ( outputUnit ) ;
```

# Demo

Daisy

# Core Audio Java

- AudioDevice object
  - start()
  - stop()
  - run()
- AudioDeviceIOProc interface
  - Basic Audio I/O callback
- Default Output Audio Unit

# Core Audio Java

- Java Doc available for API
- Should understand the fundamental designs of core audio frameworks
  - Regardless of language used
- Java API presents the same functionality

# Technology Seeding

- Ongoing development work (including some pieces that were used today) are available to developers under NDA seeding

# Resources

**Core Audio Mailing List**
Specifications and SDKs
**http://lists.apple.com**

**Developer Web Site**
**http://developer.apple.com/audio**

# Related Sessions

| | |
|---|---|
| **136 Sound & Networking for Games**<br>Enabling games with audio and networking | Room J2<br>**Fri., 2:00pm** |
| **209 Audio Processing & Sequencing Services**<br>Using Core Audio in Mac OS X | Room J2<br>**Fri., 3:30pm** |
| **210 MIDI in Mac OS X**<br>MIDI Services available in Mac OS X | Room J2<br>**Fri., 5:00pm** |

# Who to Contact

## Apple Worldwide Developer Relations

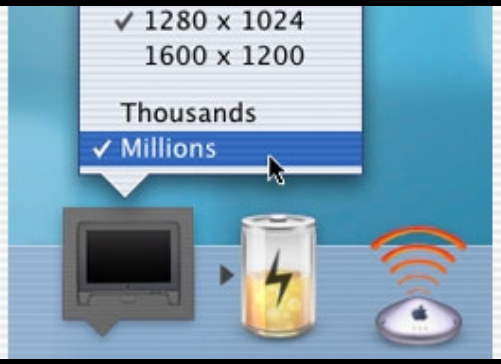
Craig Keithley
FireWire & USB Technology Manager
**keithley@apple.com**

## Developer Seeding

**audio@apple.com**

**http://developer.apple.com/events/urls.html**

# Q&A



**Craig Keithley**
USB & FireWire Technology Manager
**keithley@apple.com**

**http://developer.apple.com/events/urls.html**

WWDC
Worldwide Developers Conference 2001